PoWA Documentation

Release 3.1.0

Dalibo

Jun 15, 2017

Contents

The **PostgreSQL Workload Analyzer** is performance tool for **PostgreSQL 9.4 and superior** allowing to collect, aggregate and purge statistics on a PostgreSQL instance from various sources. It is implemented as a background worker.

This includes support for various **stat extensions**:

- *pg_stat_statements*, providing data about queries being executed
- *pg_qualstats*, providing data about predicates, or where clauses
- *pg_stat_kcache*, providing data about operating-system level cache

It supports the following extension:

- *HypoPG*, allowing you to create hypothetical indexes and test their usefulness without creating them

Additionnaly, the PoWA User Interface allows you to make the most sense of this information.

CHAPTER 1

Main components

- **PoWA-archivist** is the PostgreSQL extension, collecting statistics.
- **PoWA-web** is the graphical user interface to powa-collected metrics.
- **Stat extensions** are the actual source of data.
- **PoWA** is the whole project.

You should first take a look at the *Quickstart* guide.

Quickstart

Warning: The current version of PoWA is designed for PostgreSQL 9.4 and later. If you want to use PoWA on PostgreSQL < 9.4, please use the 1.x series

The following describes the installation of the two modules of PoWA:

- powa-archivist with the PGDG packages (RedHat/CentOS 6/7, Debian) or from the sources
- powa-web from the PGDG packages (RedHat/CentOS 7) or with python pip

Install PoWA from packages on RHEL/CentOS

Prerequisites

PoWA must be installed on the PostgreSQL instance that you are monitoring.

We suppose that you are using the packages from the PostgreSQL Development Group (https://yum.postgresql.org/ or https://apt.postgresql.org/). For example for PostgreSQL 9.6 on CentOS 7 a cluster is installed with the following commands:

```
yum install https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/
↪pgdg-centos96-9.6-3.noarch.rpm
yum install postgresql96 postgresql96-server
/usr/pgsql-9.6/bin/postgresql96-setup initdb
systemctl start postgresql-9.6
```

You will also need the PostgreSQL contrib package to provide the *pg_stat_statements* extension:

```
yum install postgresql96-contrib
```

On Debian, that would be:

```
apt-get install postgresql-9.6 postgresql-client-9.6 postgresql-contrib-9.6
```

Installation of the PostgreSQL extensions

On RedHat/CentOS, you can simply install the packages provided by the PGDG repository according to your PostgreSQL version. For example for PostgreSQL 9.6:

```
yum install powa_96 pg_qualstats96 pg_stat_kcache96 hypopg_96
```

On Debian the PoWA package exists but pg_qualstats, pg_stat_kcache and hypopg are not packaged and you will have to compile them manually *as described in the next section*:

```
apt-get install postgresql-9.6-powa
```

Once all extensions are installed or compiled, add the required modules to *shared_preload_libraries* in the *postgresql.conf* of your instance:

```
shared_preload_libraries='pg_stat_statements,powa,pg_stat_kcache,pg_qualstats'
```

Now restart PostgreSQL. Under RHEL/CentOS 6:

```
/etc/init.d/postgresql-9.6 restart
```

Under RHEL/CentOS 7:

```
systemctl restart postgresql-9.6
```

On Debian:

```
pg_ctlcluster 9.6 main restart
```

Log in to your PostgreSQL as a superuser and create a *powa* database:

```
CREATE DATABASE powa ;
```

Create the required extensions in this new database:

```
\c powa
CREATE EXTENSION pg_stat_statements;
CREATE EXTENSION btree_gist;
CREATE EXTENSION powa;
CREATE EXTENSION pg_qualstats;
CREATE EXTENSION pg_stat_kcache;
```

PoWA needs the *hypopg* extension in all databases of the cluster in order to check that the suggested indexes are efficient:

```
CREATE EXTENSION hypopg;
```

One last step is to create a role that has superuser privileges and is able to login to the cluster (use your own credentials):

```
CREATE ROLE powa SUPERUSER LOGIN PASSWORD 'astrongpassword' ;
```

The Web UI requires you to log in with a PostgreSQL role that has superuser privileges as only a superuser can access to the query text in PostgreSQL, PoWA follows the same principle.

PoWA is now up and running on the PostgreSQL-side. You still need to set up the Web interface in order to access your history. By default powa-archivist stores history for 1 day and takes a snapshot every 5 minutes. This default settings can be changed easily afterwards.

Install the Web UI

The RPM packages work for now only on RedHat/CentOS 7. For RedHat/CentOS 6 or Debian, see *the installation through pip* or *the full manual installation guide*.

You can install the web-client on any server you like. The only requirement is that the web-client can connect to the previously set-up PostgreSQL cluster.

If you're setting up PoWA on another server, you have to install the PGDG repo package again. This is required to install the *powa_96-web* package and some dependencies.

Again, for example for PostgreSQL 9.6 on CentOS 7:

```
yum install https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/
↪pgdg-centos96-9.6-3.noarch.rpm
```

Install the *powa_96-web* RPM package with its dependencies:

```
yum install powa_96-web
```

Create the */etc/powa-web.conf* config-file to tell the UI how to connect to your freshly installed PoWA database. Of course, change the given cookie to something from your own. For example to connect to the local instance on *localhost*:

```
servers={
  'main': {
    'host': 'localhost',
    'port': '5432',
    'database': 'powa'
  }
}
cookie_secret="SUPERSECRET_THAT_YOU_SHOULD_CHANGE"
```

Don't forget to allow the web server to connect to the PostgreSQL cluster, and edit your *pg_hba.conf* accordingly.

Then, run powa-web:

```
powa-web
```

The Web UI is now available on port 8888, for example on http://localhost:8888/. You may have to configure your firewall to open the access to the outside. Use the role created earlier in PostgreSQL to connect to the UI.

Build and install powa-archivist from the sources

Prerequisites

You will need a compiler, the appropriate PostgreSQL development packages, and some contrib modules.

While on most installation, the contrib modules are installed with a postgresql-contrib package, if you wish to install them from source, you should note that only the following modules are required:

- btree_gist
- pg_stat_statements

On RedHat/CentOS:

```
yum install postgresql96-devel postgresql96-contrib
```

On Debian:

```
apt-get install postgresql-server-dev-9.6 postgresql-contrib-9.6
```

Installation

Download powa-archivist latest release:

```
wget https://github.com/dalibo/powa-archivist/archive/REL_3_1_0.tar.gz
```

A convenience script is offered to build every project that PoWA can take advantage of:

```
#!/bin/bash
# This script is meant to install every PostgreSQL extension compatible with
# PoWA.
wget https://github.com/dalibo/pg_qualstats/archive/1.0.2.tar.gz -O pg_
↪qualstats-1.0.2.tar.gz
tar zxvf pg_qualstats-1.0.2.tar.gz
cd pg_qualstats-1.0.2
(make && sudo make install)  > /dev/null 2>&1
cd ..
rm pg_qualstats-1.0.2.tar.gz
rm pg_qualstats-1.0.2 -rf
wget https://github.com/dalibo/pg_stat_kcache/archive/REL2_0_3.tar.gz -O pg_
↪stat_kcache-REL2_0_3.tar.gz
tar zxvf pg_stat_kcache-REL2_0_3.tar.gz
cd pg_stat_kcache-REL2_0_3
(make && sudo make install)  > /dev/null 2>&1
cd ..
rm pg_stat_kcache-REL2_0_3.tar.gz
rm pg_stat_kcache-REL2_0_3 -rf
(make && sudo make install)  > /dev/null 2>&1
echo ""
echo "You should add the following line to your postgresql.conf:"
echo ''
echo "shared_preload_libraries='pg_stat_statements,powa,pg_stat_kcache,pg_
↪qualstats'"
echo ""
echo "Once done, restart your postgresql server and run the install_all.sql␣
↪file"
```

```
echo "with a superuser, for example: "
echo "  psql -U postgres -f install_all.sql"
```

This script will ask for your super user password, provided the sudo command is available, and install powa, pg_qualstats and pg_stat_kcache for you.

> **Warning:** This script is not intended to be run on a production server, as it will install the development version of each extension and not the latest stable release. It has been removed since the 2.0.1 release of PoWA.

Once done, you should modify your PostgreSQL configuration as mentioned by the script, putting the following line in your *postgresql.conf* file:

```
shared_preload_libraries='pg_stat_statements,powa,pg_stat_kcache,pg_qualstats'
```

Optionally, you can install the hypopg extension the same way from https://github.com/dalibo/hypopg/releases.

And restart your server, according to your distribution's preferred way of doing so, for example:

Init scripts:

```
/etc/init.d/postgresql-9.6 restart
```

Debian pg_ctlcluster wrapper:

```
pg_ctlcluster 9.6 main restart
```

Systemd:

```
systemctl restart postgresql
```

The last step is to create a database dedicated to the PoWA repository, and create every extension in it. The install_all.sql file performs this task:

```
psql -U postgres -f install_all.sql
CREATE DATABASE
You are now connected to database "powa" as user "postgres".
CREATE EXTENSION
CREATE EXTENSION
CREATE EXTENSION
CREATE EXTENSION
CREATE EXTENSION
```

Install powa-web anywhere

You do not have to install the GUI on the same machine your instance is running.

Prerequisites

- The Python language, either 2.6, 2.7 or > 3
- The Python language headers, either 2.6, 2.7 or > 3
- The pip installer for Python. It is usually packaged as "python-pip", for example:

Debian:

```
sudo apt-get install python-pip python-dev
```

RedHat/CentOS:

```
sudo yum install python-pip python-devel
```

Installation

To install powa-web, just issue the following comamnd:

```
sudo pip install powa-web
```

Then you'll have to configure a config file somewhere, in one of those location:

- /etc/powa-web.conf
- ~/.config/powa-web.conf
- ~/.powa-web.conf
- ./powa-web.conf

The configuration file is a simple JSON one. Copy the following content to one of the above locations:

```
servers={
  'main': {
    'host': 'localhost',
    'port': '5432',
    'database': 'powa'
  }
}
cookie_secret="SUPERSECRET_THAT_YOU_SHOULD_CHANGE"
```

The servers key define a list of server available for connection by PoWA-web. You should ensure that the pg_hba.conf file is properly configured.

The cookie_secret is used as a key to crypt cookies between the client and the server. You should DEFINETLY not keep the default if you value your security.

Other options are described in the full documentation.

Then, run powa-web:

```
powa-web
```

The UI is now available on the 8888 port. Login with the credentials of the *powa* PostgreSQL user.

Frequently Asked question

Some queries don't show up in the UI

That's a know limitation with the current implementation of powa-web.

For now, the UI will only display information about queries that have been run on **at least** two distinct snapshots of powa-archivist (parameter *powa.frequency*).

This is however usually not a problem since queries only executed a few time and never again are not really a target for optimization.

Security

Warning: You need to be careful about the security of your PostgreSQL instance when installing PoWA.

We designed POWA so that the user interface will only communicate with PostgreSQL via prepared statements. This will prevent the risk of SQL injection.

However to connect to the PoWA User Interface, you will use the login and password of a PostgreSQL user. If you don't protect your communications, an attacker placed between the GUI and PostgreSQL, or between you and the GUI, could gain your user rights to your database server.

Therefore we **strongly** recommend the following precautions:

- Read the Great PostgreSQL Documentation
- Check your *pg_hba.conf* file
- Do not allow users to access PoWA from the Internet
- Do not allow users to access PostgreSQL from the Internet
- Run PoWA on a HTTPS server and disable HTTP access
- Use SSL to protect the connection between the GUI and PostgreSQL
- Reject unprotected connections between the GUI and PostgreSQL (*hostnossl .... reject*)
- Check your *pg_hba.conf* file again

Please also note that you need to manually authorize the roles to see the data in the powa database. For instance, you might run:

```
powa=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO ui_user;
powa=# GRANT SELECT ON pg_statistic TO ui_user;
```

User objects

powa-web will connect to the databases you select to help you optimize them.

Therefore, for each postgres roles using powa, you also need to:

- grant **SELECT** privilege on the pg_statistic and the user tables (don't forget tables that aren't in the public schema).
- give **CONNECT** privilege on the databases.

If you don't, some useful parts of the UI won't work as intended.

PoWA archivist

Installation

Prerequisites

- PostgreSQL >= 9.4
- PostgreSQL contrib modules (pg_stat_statements and btree_gist)
- PostgreSQL server headers

On Debian, the PostgreSQL server headers are installed via the `postgresql-server-dev-X.Y` package:

```
apt-get install postgresql-server-dev-9.4 postgresql-contrib-9.4
```

On RPM-based distros:

```
yum install postgresql94-devel postgresql94-contrib
```

You also need a C compiler and other standard development tools.

On Debian, these can be installed via the `build-essential` package:

```
apt-get install build-essential
```

On RPM-based distros, the "Development Tools" can be used:

```
yum groupinstall "Development Tools"
```

Installation

Grab the latest release, and install it:

```
wget https://github.com/dalibo/powa-archivist/archive/REL_3_1_0.tar.gz -O␣
↪powa-archivist-REL_3_1_0.tar.gz
tar zxvf powa-archivist-REL_3_1_0.tar.gz
cd powa-archivist-REL_3_1_0
```

Compile and install it:

```
make
sudo make install
```

It should output something like the following :

```
/bin/mkdir -p '/usr/share/postgresql-9.4/extension'
/bin/mkdir -p '/usr/share/postgresql-9.4/extension'
/bin/mkdir -p '/usr/lib64/postgresql-9.4/lib64'
/bin/mkdir -p '/usr/share/doc/postgresql-9.4/extension'
/usr/bin/install -c -m 644 powa.control '/usr/share/postgresql-9.4/extension/'
/usr/bin/install -c -m 644 powa--2.0.sql '/usr/share/postgresql-9.4/extension/'
/usr/bin/install -c -m 644 README.md '/usr/share/doc/postgresql-9.4/extension/'
/usr/bin/install -c -m 755  powa.so '/usr/lib64/postgresql-9.4/lib64/'
```

Create the PoWA database and create the required extensions, with the following statements:

```
CREATE EXTENSION pg_stat_statements;
CREATE EXTENSION btree_gist;
CREATE EXTENSION powa;
```

Example:

```
bash-4.1$ psql
psql (9.3.5)
Type "help" for help.
postgres=# create database powa;
CREATE DATABASE
postgres=# \c powa
You are now connected to database "powa" as user "postgres".
powa=# create extension pg_stat_statements ;
CREATE EXTENSION
powa=# create extension btree_gist ;
CREATE EXTENSION
powa=# create extension powa;
CREATE EXTENSION
```

As PoWA-archivist is implemented as a background worker, the library must be loaded at server start time.

For this, modify the `postgresql.conf` configuration file, and add powa and pg_stat_statements to the `shared_preload_libraries` parameter:

```
shared_preload_libraries = 'pg_stat_statements,powa'
```

If possible, activate `track_io_timing` too:

```
track_io_timing = on
```

PostgreSQL should then be restarted.

Configuration

The following configuration parameters (GUCs) are available in `postgresql.conf`:

powa.frequency: Defaults to `5min`. Defines the frequency of the snapshots, in milliseconds or any time unit supported by PostgreSQL. Minimum 5s. You can use the usual postgresql time abbreviations. If not specified, the unit is seconds. Setting it to -1 will disable powa (powa will still start, but it won't collect anything anymore, and wont connect to the database).

powa.retention: Defaults to `1d` (1 day) Automatically purge data older than that. If not specified, the unit is minutes.

powa.database: Defaults to `powa` Defines the database of the workload repository.

powa.coalesce: Defaults to `100`. Defines the amount of records to group together in the table.

Integrating another stat extension in Powa

Clone the repository:

```
git clone https://github.com/dalibo/powa-archivist/
cd powa-archivist/
make && sudo make install
```

Any modification to the background-worker code will need a PostgreSQL restart.

In order to contribute another source of data, you will have to implement the following functions:

snapshot: This function is responsible for taking a snapshot of the data source data, and store it somewhere. Usually, this is done in a staging table named **powa_my_data_source_history_current**. It will be called every *powa.frequency* seconds. The function signature looks like this:

```
CREATE OR REPLACE FUNCTION powa_my_data_source_snapshot() RETURNS void AS $PROC$
...
$PROC$ language plpgsql;
```

aggregate: This function will be called after every *powa.coalesce* number of snapshots. It is responsible for aggregating the current staging values into another table, to reduce the disk usage for PoWA. Usually, this will be done in an aggregation table named **powa_my_data_source_history**. The function signature looks like this:

```
CREATE OR REPLACE FUNCTION powa_my_data_source_aggregate() RETURNS void AS $PROC$
...
$PROC$ language plpgsql;
```

purge: This function will be called after every 10 aggregates and is responsible for purging stale data that should not be kept. The function should take the *powa.retention* global parameter into account to prevent removing data that would still be valid.

```
CREATE OR REPLACE FUNCTION powa_my_data_source_aggregate() RETURNS void AS $PROC$
...
$PROC$ language plpgsql;
```

unregister: This function will be called if the related extension is dropped.

Please note that the **module** name used in the **powa_functions** table has to be the same as the extension name, otherwise the function will not be called.

This function should at least remove entries from **powa_functions** table. A minimal function would look like this:

```
CREATE OR REPLACE function public.powa_my_data_source_unregister() RETURNS bool AS
$_$
BEGIN
    DELETE FROM public.powa_functions WHERE module = 'my_data_source';
        RETURN true;
END;
$_$
language plpgsql;
```

Each of these functions should then be registered:

```
INSERT INTO powa_functions (module, operation, function_name, added_manually)
VALUES  ('my_data_source', 'snapshot',   'powa_mydatasource_snapshot',   true),
        ('my_data_source', 'aggregate',  'powa_mydatasource_aggregate',  true),
        ('my_data_source', 'unregister', 'powa_mydatasource_unregister', true),
        ('my_data_source', 'purge',      'powa_mydatasource_purge',      true);
```

PoWA-web

Installation

You can install PoWA-web either using pip or manually.

On Centos 6, you can avoid installing the header files for Python and PostgreSQL by using the package for psycopg2:

```
yum install python-pip python-psycopg2
pip install powa-web
```

Manual install

You'll need the following dependencies:

- python 2.6, 2.7 or > 3
- psycopg2
- sqlalchemy >= 0.8.0
- tornado >= 2.0

debian

```
apt-get install python python-psycopg2 python-sqlalchemy python-tornado
```

archlinux

```
pacman -S python python-psycopg2 python-sqlalchemy python-tornado
```

fedora

```
TODO
```

Then, download the latest release on pypi, uncompress it, and copy the sample configuration file:

```
wget https://pypi.io/packages/source/p/powa-web/powa-web-3.1.3.tar.gz
tar -zxvf powa-web-3.1.3.tar.gz
cd powa-web-3.1.3
cp ./powa-web.conf-dist ./powa-web.conf
./powa-web
```

Then, jump on the next section to configure powa-web.

Configuration

The powa-web configuration is stored as a simple python file. Powa-web will search its config as either of these files, in this order:

- /etc/powa-web.conf
- ~/.config/powa-web.conf
- ~/.powa-web.conf
- ./powa-web.conf

You'll then be noticed of the address and port on which the UI is available. The default is 0.0.0.0:888, as indicated in this message:

- [I 161105 20:27:39 powa-web:12] Starting powa-web on 0.0.0.0:8888

The following options are required:

servers (dict): A dictionary mapping server names to connection information.

```
servers={
  'main': {
    'host': 'localhost',
    'port': '5432',
    'database': 'powa'
  }
}
```

Warning:

If any of your databases is not in **utf8** encoding, you should specify a client_encoding option as shown below. This requires at least psycopg2 version 2.4.3

```
servers={
  'main': {
    'host': 'localhost',
    'port': '5432',
    'database': 'powa',
    'query': {'client_encoding': 'utf8'}
  }
}
```

Note:

You can set a username and password to allow logging into powa-web without providing credentials. In this case, the powa-web.conf file must be modified like this:

```
servers={
  'main': {
    'host': 'localhost',
    'port': '5432',
    'database': 'powa',
    'username' : 'pg_username',
    'password' : 'the password',
    'query': {'client_encoding': 'utf8'}
  }
}
```

cookie_secret (str): A secret key used to secure cookies transiting between the web browser and the server.

```
cookie_secret="SECRET_STRING"
```

The following options are optional:

port (int): The port on which the UI will be available (default 8888)

address (str): The IP address on which the UI will be available (default 0.0.0.0)

See also:

Deployment Options

Apache

PoWA can easily be deployed using Apache mod_wsgi module.

First you have to install and configure Powa like in the *quickstart* section. Check that the powa-web executable works before proceeding.

In your apache configuration file, you should:

- load the mod_wsgi module
- configure it.

The various python3.4 version in the paths below should be set your actual python version:

```
LoadModule wsgi_module modules/mod_wsgi.so
<VirtualHost *:80>
  ServerName myserver.example.com

  DocumentRoot /var/www/

  ErrorLog /var/log/httpd/powa.error.log
  CustomLog /var/log/httpd/powa.access.log combined

  WSGIScriptAlias / /usr/lib/python3.4/site-packages/powa/powa.wsgi

  Alias /static /usr/lib/python3.4/site-packages/powa/static/
</VirtualHost>
```

Development

This page acts as a central hub for resources useful for PoWA developers.

PoWA-Web

This section only covers the most simple changes one would want to make to PoWA. For more comprehensive documentation, see the Powa-Web project documentation itself.

Clone the repository:

```
git clone https://github.com/dalibo/powa-web/
cd powa/
make && sudo make install
```

To run the application, use run_powa.py, which will run powa in debug mode. That means the javascript files will not be minified, and will not be compiled into one giant source file.

CSS files are generated using *sass <http://sass-lang.com>*. Javascript files are splitted into AMD modules, which are managed by *requirejs <http://requirejs.org/>* and compiled using *grunt <http://gruntjs.com>*.

These projects depend on NodeJS, and NPM, its package manager, so make sure you are able to install them on your distribution.

Install the development dependencies:

```
npm install -g grunt-cli
npm install .
```

Then, you can run `grunt` to update only the css files, or regenerate optimized javascript builds with `grunt dist`.

Stats Extensions

The PoWA-archivist collects data from various stats extensions. To be used in PoWA, a stat extensions has to expose a number of PL/pgSQL functions as stated in *Integrating another stat extension in Powa*.

Currently, the list of supported stat extensions is as follows:

pg_stat_statements

The pg_stat_statements extension records statistics of all SQL queries (aka "statements") executed on a given PostgreSQL server.

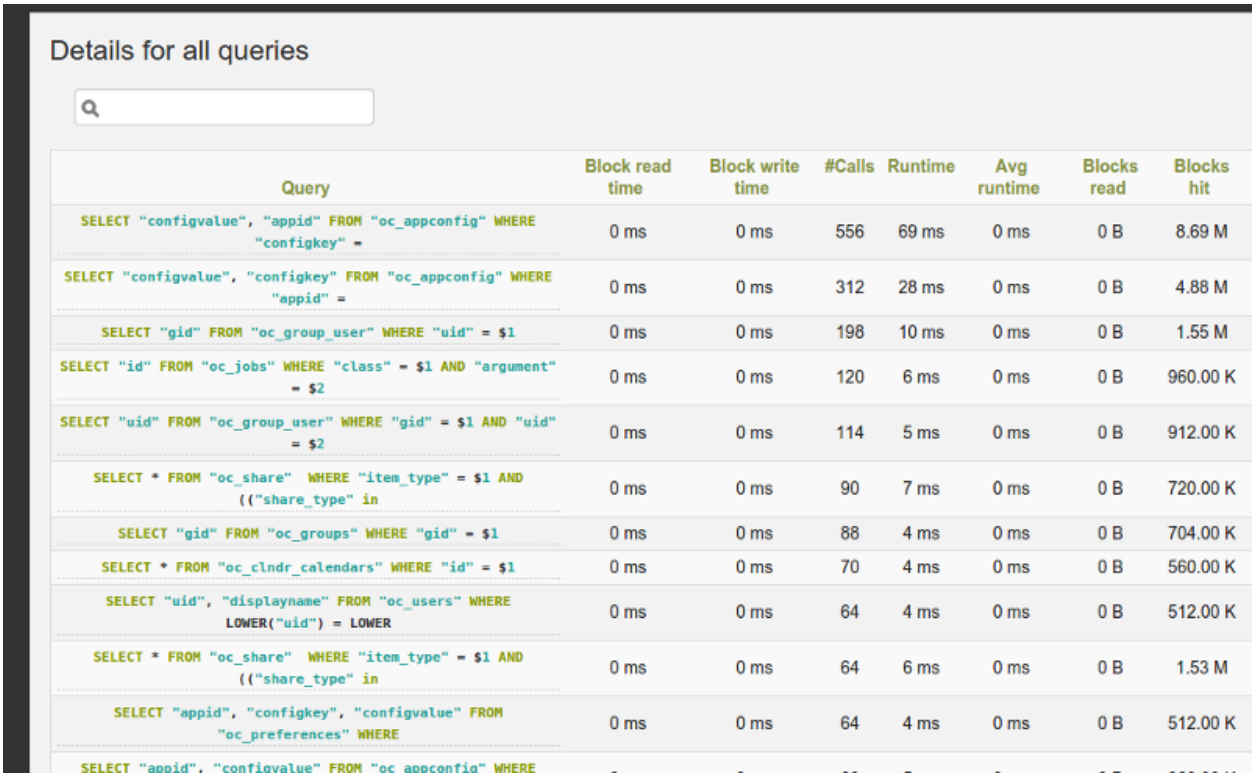
The statistics gathered are available in view called `pg_stat_statements`. This view contains one row for each distinct database ID, user ID and query ID. However the number of distinct statements tracked cannot exceed a certain limit (5 000 by default)

The pg_stat_statements extension is a key component of the PoWA Suite, installing it is **mandatory**.

Where is it used in powa-web ?

The PoWA user interface (`powa-web`) relies heavily on pg_stat_statements, so you'll see it used in almost every screen of the tool.

The most useful feature is probably the "Query details" chart which show advanced statistics for each SQL query.



Installation

pg_stat_statements is an official extension and it is released along with other extensions in the official PostgreSQL packages. You will find it in the `contrib` folder. Depending on which Operating System, you're using you may need to install a separate package to use it. For instance, on `debian` you may need to install the `postgresql-contrib` package.

Then you just have to declare the extension in the `postgresql.conf` file, like this :

```
shared_preload_libraries = 'pg_stat_statements'
```

Restart the PostgreSQL server to reload the libraries.

Connect to the server as a superuser and type:

```
CREATE EXTENSION pg_stat_statements
```

Configuration

There's a few parameters that you can add to the `postgresql.conf`. For instance you can increase the track limit and allow PostgreSQL to record 10 000 distinct queries:

```
pg_stat_statements.max = 10000
```

For more information about the pg_stat_statements, please read the PostgreSQL documentation:

http://www.postgresql.org/docs/current/static/pgstatstatements.html

Examples

See Also

- http://www.craigkerstiens.com/2013/01/10/more-on-postgres-performance/

pg_qualstats

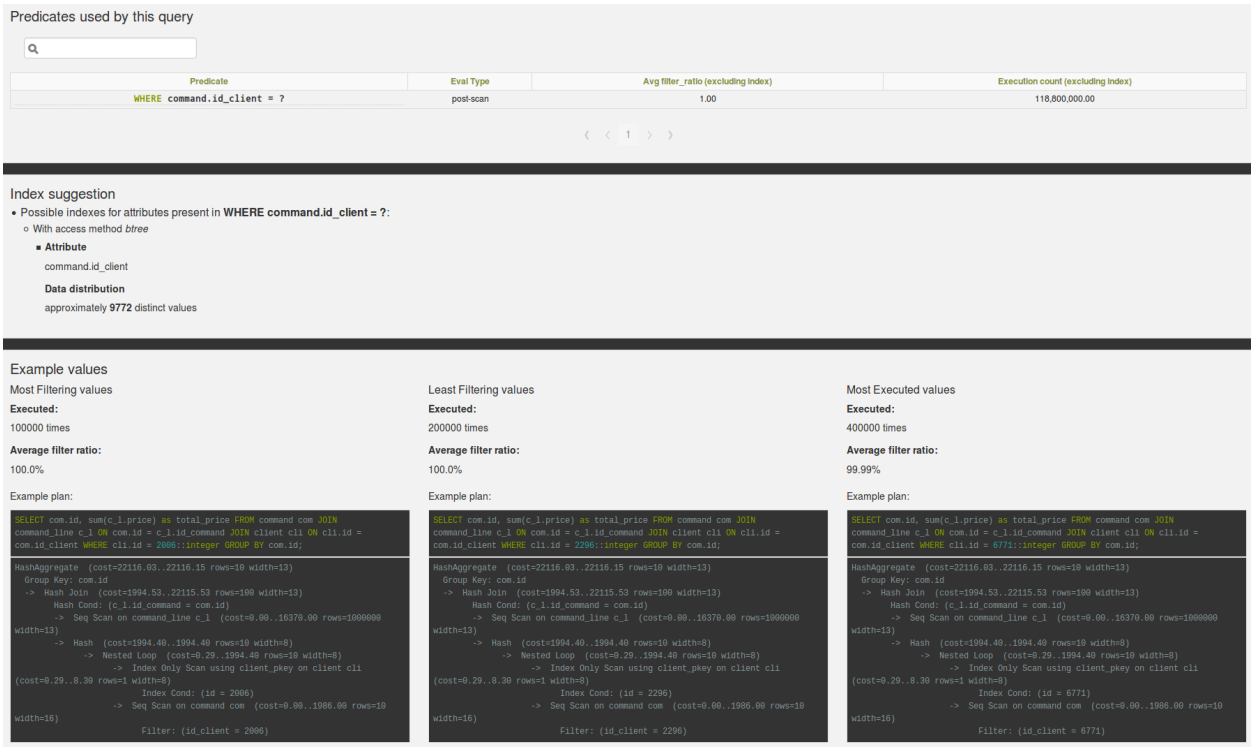
pg_qualstats is a PostgreSQL extension keeping statistics on predicates found in `` `WHERE` `` statements and `` `JOIN` `` clauses.

The goal of this extension is to allow the DBA to answer some specific questions, whose answers are quite hard to come by:

- what is the set of queries using this column ?
- what are the values this where clause is most often using ?
- do I have some significant skew in the distribution of the number of returned rows if use some value instead of one another ?
- which columns are often used together in a WHERE clause ?

Where is it used in powa-web ?

If the extension is available, you should see a "list of quals" table on the query page, as well as explain plans for your query and a list of index suggestions:



From this list, you can then go on to the per-qual page.

Installation

As seen in *Quickstart*, the PostgreSQL development packages should be available.

First, download and extract the latest release of *pg_qualstats*:

```
wget https://github.com/dalibo/pg_qualstats/archive/1.0.2.tar.gz -O pg_
↪qualstats-1.0.2.tar.gz
tar zxvf pg_qualstats-1.0.2.tar.gz
cd pg_qualstats-1.0.2
```

Then, compile the extension:

```
make
```

Then install the compiled package:

```
make install
```

Then you just have to declare the extension in the `postgresql.conf` file, like this :

```
shared_preload_libraries = 'pg_stat_statements,pg_qualstats'
```

Restart the PostgreSQL server to reload the libraries.

Connect to the server as a superuser and type:

```
CREATE EXTENSION pg_qualstats;
```

Using with PoWA

If you want PoWA to handle this extension, you have to connect as a superuser on the database where you installed PoWA, and type:

```
SELECT powa_qualstats_register();
```

Configuration

The following configuration parameters are available, in postgresql.conf:

pg_qualstats.enabled: Defaults to `true`. Enable pg_qualstats. Can be useful if you want to enable / disable it without restarting the server.

pg_qualstats.max: Defaults to `1000`. Number of entries to keep. As a rule of thumb, you should keep at least `pg_stat_statements.max` entries if `pg_qualstats.track_constants` is disabled, else it should be roughly equal to the number of queries executed during `powa.frequency` interval of time.

pg_qualstats.track_pg_catalog: Defaults to `false`. Determine if predicates on pg_catalog tables should be tracked too.

pg_qualstats.resolve_oids: Defaults to `false`. Determine if during predicates collection, the actual name of the objects should be stored alongside their OIDs. The overhead is quite non-negligible, since each entry will occupy 616 bytes instead of 168.

pg_qualstats.track_constants: Defaults to `true`. If true, each new value for each predicate will result in a new entry. Eg, `WHERE id = 3` and `WHERE id = 4` will results in two entries in pg_qualstats. If disabled, only one entry for `WHERE id = ?` will be kept. Turning this off drastically reduces the number of entries to keep, at the price of not getting any hindsight on most frequently used values.

pg_qualstats.sample_rate: (Used to be "sample_ratio") Defaults to `1`, which means `1 / MAX_CONNECTIONS` The ratio of queries that should be sampled. 1 means sample every single query, 0 basically deactivates the feature, and -1 is automatically sized to `1/ MAX_CONNECTIONS`. For example, a sample_rate of `0.1` would mean one of out ten queries should be sampled.

SQL Objects

The extension defines the following objects:

function`pg_qualstats_reset()`
Resets statistics gathered by pg_qualstats.

function`pg_qualstats()`

Returns: A SETOF record containing the data gathered by pg_qualstats

Attributes:

userid (oid): the user who executed the query

dbid (oid): the database on which the query was executed

lrelid (oid): oid of the relation on the left hand side

lattnum (attnum): attribute number of the column on the left hand side

opno (oid): oid of the operator used in the expression

rrelid (oid): oid of the relation on the right hand side

rattnum (attnum): attribute number of the column on the right hand side

uniquequalnodeid(bigint): hash of the parent `AND` expression, if any. This is useful for identifying predicates which are used together.

qualnodeid(bigint): the predicate hash. Everything (down to constants) is used to compute this hash

occurences (bigint): the number of times this predicate has been seen

execution_count (bigint): the total number of execution of this predicate.

nbfiltered (bigint): the number of lines filtered by this predicate

constant_position (int): the position of the constant in the original query, as filled by the lexer.

queryid (oid): the queryid identifying this query, as generated by pg_stat_statements

constvalue (varchar): a string representation of the right-hand side constant, if any, truncated to 80 bytes.

eval_type (char): the evaluation type. Possible values are `f` for execution as a filter (ie, after a Scan) or `i` if it was evaluated as an index predicate. If the qual is evaluated as an index predicate, then the nbfiltered value will most likely be 0, except if there was any rechecked conditions.

Example:

```
powa=# select * from powa_statements where queryid != 2;
powa=# select * from pg_qualstats();
-[ RECORD 1 ]-----+-----------
userid            | 16384
dbid              | 850774
lrelid            | 851367
lattnum           | 1
opno              | 417
rrelid            |
rattnum           |
qualid            |
uniquequalid      |
qualnodeid        | 1711571257
uniquequalnodeid  | 466568149
```

occurences | 1 execution_count | 1206 nbfiltered | 0 constant_position | 47 queryid | 3644521490 constvalue | 2::integer eval_type | f

function pg_qualstats_names()

This function is the same as pg_qualstats, but with additional columns corresponding to the resolved names, if `pg_qualstats.resolve_oids` is set to `true`.

Returns: The same set of columns than *pg_qualstats()*, plus the following ones:

rolname (text): the name of the role executing the query. Corresponds to userid.

dbname (text): the name of the database on which the query was executed. Corresponds to dbid.

lrelname (text): the name of the relation on the left-hand side of the qual. Corresponds to lrelid.

lattname (text): the name of the attribute (column) on the left-hand side of the qual. Corresponds to rrelid.

opname (text): the name of the operator. Corresponds to opno.

viewpg_qualstats
This view is just a simple wrapper on the *pg_qualstats()* function, filtering on the current database for convenience.

viewpg_qualstats_pretty
This view resolves oid "on the fly", for the current database.

Returns:

left_schema (name): the name of the left-hand side relation's schema.

left_table (name): the name of the left-hand side relation.

left_column (name): the name of the left-hand side attribute.

operator (name): the name of the operator.

right_schema (name): the name of the right-hand side relation's schema.

right_table (name): the name of the right-hand side relation.

right_column (name): the name of the operator.

execution_count (bigint): the total number of time this qual was executed.

nbfiltered (bigint): the total number of tuples filtered by this qual.

typequal
Attributes:

relid (oid): the relation oid

attnum (integer): the attribute number

opno (oid): the operator oid

eval_type (char): the evaluation type. See *pg_qualstats()* for an explanation of the eval_type.

typequalname
Pendant of *qual*, but with names instead of oids

Attributes:

relname (text): the relation oid

attname (text): the attribute number

opname (text): the operator name

eval_type (char): the evaluation type. See *pg_qualstats()* for an explanation of the eval_type.

pg_stat_kcache

pg_stat_kcache is a PostgreSQL extension gathering statistics on system metrics.

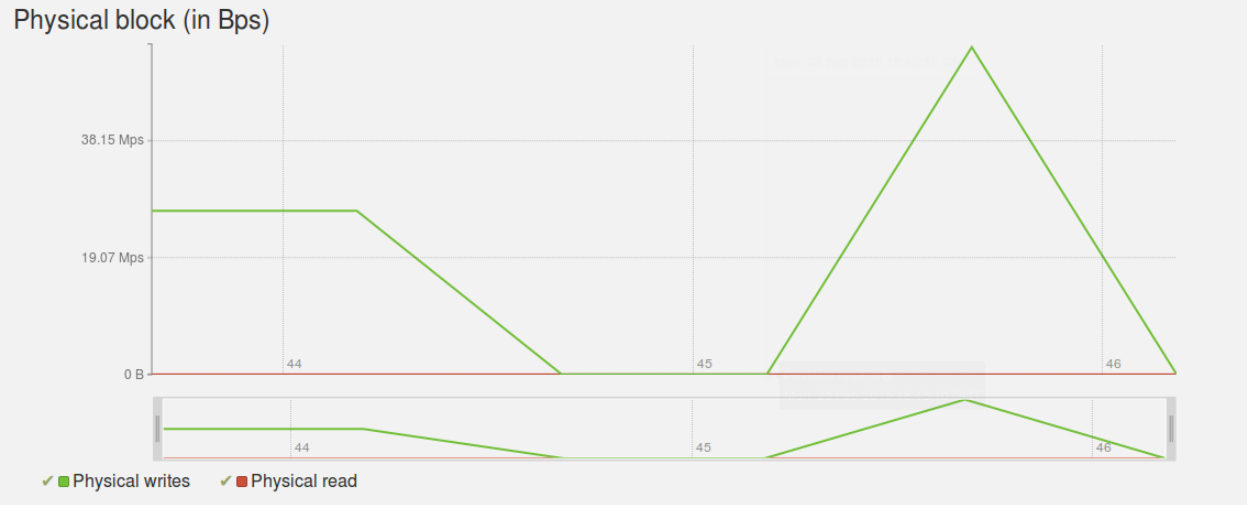
Thanks to this extension, the DBA can see how much resource each query, user and/or database is consuming. The resources are:

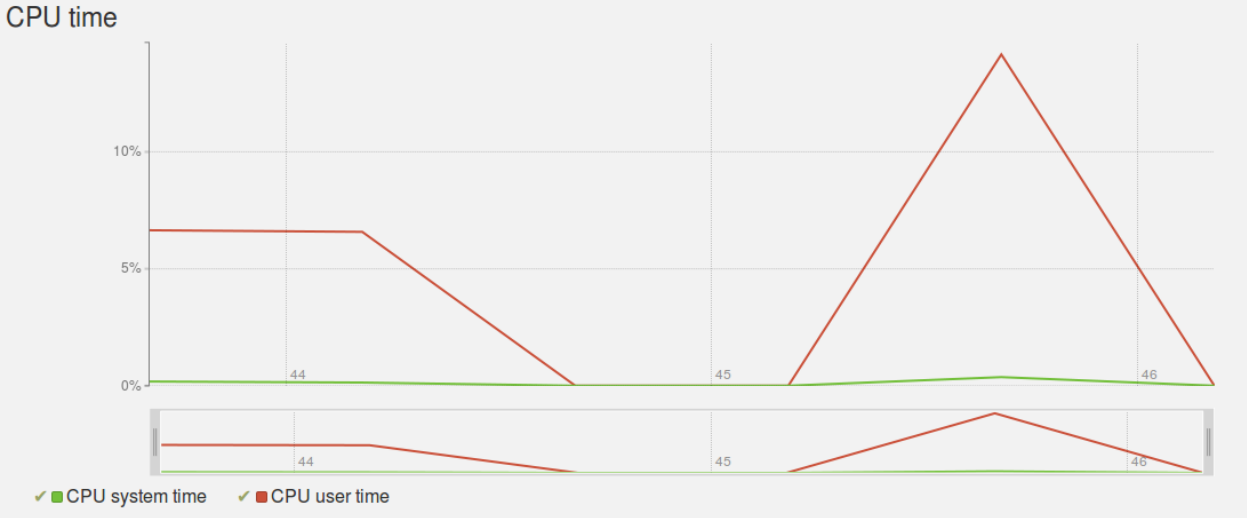
- CPU (user time and system time)
- Physical disk access (read and write)

Physical disk access are essential in calculating a real hit ratio (cached_reads/all_reads). Without this, we only have the shared_buffers' hit ratio, and some of the reads made by Postgres could be served by the system cache.

Where is it used in powa-web ?

If the extension is available, you should see "Physical block" and "CPU time" graphs on the query page:





The **CPU time** metrics indicate the percentage of query runtime spent consuming either *user cpu time* or *system cpu time*.

The "Hit ratio" graph will also handle this extension, displaying the following metrics :

- **Shared buffers hit ratio**: percentage of blocks read from shared buffers (memory)
- **System cache hit ratio**: precentage of blocks read from the system cache (memory)
- **Disk hit ratio**: Percentage of blocks which needed a physical disk read

Installation

pg_stat_kcache should work with any POSIX operating system. Therefore, it won't on Windows.

As seen in *Quickstart*, the PostgreSQL development packages should be available.

First, you need to download and extract the lastest release of pg_stat_kcache.

```
wget https://github.com/dalibo/pg_stat_kcache/archive/REL2_0_3.tar.gz -O pg_
↪stat_kcache-REL2_0_3.tar.gz
tar zxvf pg_stat_kcache-REL2_0_3.tar.gz
cd pg_stat_kcache-REL2_0_3
```

Then, compile the extension:

```
make
```

If everything goes fine, you will have this kind of output :

```
gcc -O0 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -
↪Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing -
↪fwrapv -fexcess-precision=standard -g -fpic -I. -I./ -I/home/rjuju/postgres/pgs/
↪postgresql-9.4.beta2/include/server -I/home/rjuju/postgres/pgs/postgresql-9.4.beta2/
↪include/internal -D_GNU_SOURCE -I/usr/include/libxml2   -c -o pg_stat_kcache.o pg_
↪stat_kcache.c
gcc -O0 -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -
↪Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing -
↪fwrapv -fexcess-precision=standard -g -fpic -shared -o pg_stat_kcache.so pg_stat_
↪kcache.o -L/home/rjuju/postgres/pgs/postgresql-9.4.beta2/lib -L/usr/lib/x86_64-
↪linux-gnu  -Wl,--as-needed -Wl,-rpath,'/home/rjuju/postgres/pgs/postgresql-9.4.
↪beta2/lib',--enable-new-dtags
```

Then install the compiled file. This step has to be made with the user that has installed PostgreSQL. If you have used a package, it will be certainly be root. If so:

```
sudo make install
```

Else, sudo into the user that owns your PostgreSQL executables, and

```
make install
```

Then you just have to declare the extension in the `postgresql.conf` file, like this :

```
shared_preload_libraries = 'pg_stat_statements,pg_stat_kcache'
```

Restart the PostgreSQL server to reload the libraries.

Connect to the server as a superuser and type:

```
CREATE EXTENSION pg_stat_kcache ;
```

Using with PoWA

If you want PoWA to handle this extension, you have to connect as a superuser on the database where you installed PoWA, and type:

```
SELECT powa_kcache_register();
```

Configuration

pg_stat_kcache will retain as many query statistic as pg_stat_statements, so there's nothing to configure.

Examples

See Also

- *pg_stat_statements*
- *pg_qualstats*

HypoPG

HypoPG isn't a stat extension, but it's a useful extension to take full advantage of all the PoWA features.

HypoPG allows you to create hypothetical indexe. A hypothetical index is an index that doesn't exists on disk. It's therefore almost instant to create and doesn't add any IO cost, whether at creation time or at maintenance time. The goal is obviously to check if an index is useful before spending too much time, I/O and disk space to create it.

With this extension, you can create hypothetical indexes, and then with EXPLAIN check if PostgreSQL would use them or not.

Except the hypopg extension, which have to be install in each database, pg_stat_statements, pg_qualstats and pg_stat_kcache just have to be install on 'powa' database.

Impact on performances

Using PoWA will have a small negative impact on your PostgreSQL server performances. It is hard to evaluate precisely this impact but we can analyze it in 3 parts :

- First of all, you need to activate the pg_stat_statements module. This module itself may slow down your instance, but some benchmarks (for example, here and here) show that the impact is not that big.
- Second, the PoWA collector should have a very low impact, but of course that depends on the frequency at which you collect data. If you do it every 5 seconds, you'll definitely see something. At 5 minutes, the impact should be minimal.
- And finally the POWA GUI will have an impact too if you run it on the PostgreSQL instance, but it really depends on many user will have access to it.

All in all, we strongly feel that the performance impact of POWA is nothing compared to being in the dark and not knowing what is running on your database. And in most cases the impact is lower than setting `log_min_duration_statement = 0`.

See our own benchmark for more details:

- PoWA vs The Badger

Support

Community Support

You can find help, news and security alerts on the opm-users mailing list :

https://groups.google.com/forum/?hl=fr#!forum/powa-users

You can also join directly the developer team on the #powa channel of the freenode IRC network

To report an issue, please use the bug tracking system in the github project page: https://github.com/dalibo/powa

Commercial Support

DALIBO, as the main sponsor of the project, can provide enterprise-grade support services for both PostgreSQL and OPM. See http://www.dalibo.com for more details.

Release Notes

What's new in PoWA 3.0.0

December 7, 2015

Better predicate analyzer

The pg_qualstats (https://github.com/dalibo/pg_qualstats) extension stores new counters. It's now possible to know the most executed predicates in relation to all the related queries. It also tracks non-normalized queries so that it's possible to execute an EXPLAIN of any query tracked by pg_stat_statements.

Database global optimization

PoWA is now able to use statistics about every predicate used by any query executed on a database to suggest the smallest index set that optimizes every one of those predicates.

In particular, the heuristics place heavy emphasis in consolidating many indexes into one by giving preference to definitions spanning multiple columns. This can provide new information about the actual load and correlation between predicates that are traditionally hard to discover for the DBA.

Index suggestion check

Thanks to the HypoPG (https://github.com/dalibo/hypoopg) extension, the benefits of the suggested index creations can automatically be checked by running the queries against hypothetical indexes. You can see instantly if the suggested index is relevant and how much it'll improve the query.

Documentation

- Complete user documentation available at http://powa.readthedocs.io/

Backward compatibility

- PoWA 2.0 and later is NOT COMPATIBLE with PostgreSQL 9.3. If you're using PoWA with PostgreSQL 9.3, you can either keep PoWA 1.2 or upgrade to PostgreSQL 9.4 and switch to PoWA 3.0.0.

What's new in PoWA 2.0

March 2, 2015

New User Interface

- The web interface is now a seprate module called `powa-web`
- Complete rewite of the previous HTML UI
- We dropped mojolicious and use Tornado instead
- New Bar Graph
- New configuration view
- New index suggestion widget
- New physical resource consumption graphs
- Pie Charts
- Histogramm for qual constants values
- Better Global Query Chart
- Breadcrumbs
- Check PoWA installation on login
- Python 2.6, 2.7 and 3.4 compatibility

New Stat sources

- The core engine is now a seprate module called `powa-archivist`
- Integration of pg_qualstats
- Integration of pg_stat_kcache

Documentation

- Complete user documentation available at http://powa.readthedocs.io/

Backward compatibility

- PoWA 2.0 and later is NOT COMPATIBLE with PostgreSQL 9.3. If you're using PoWA with PostgreSQL 9.3, you can either keep PoWA 1.2 or upgrade to PostgreSQL 9.4 and switch to PoWA 2.0.

Contributing

POWA is an open project available under the PostgreSQL License. Any contribution to build a better tool is welcome.

Talk

If you have ideas or feature requests, please post them to our mailing list here: https://groups.google.com/forum/?hl=fr#!forum/powa-users

Test

If you've found a bug, please tell us more here : https://github.com/dalibo/powa/issues

Code

For a better modularity, the code base is split on 3 separate github repositories:

- main repo and doc : https://github.com/dalibo/powa
- core engine : https://github.com/dalibo/powa-archivist
- use interface : https://github.com/dalibo/powa-web

Index